UNITED STATES DESIGN PATENT APPLICATION

FOR

MECHANISM TO CONTROL HARDWARE INTERRUPT ACKNOWLEDGEMENT IN A VIRTAL MACHINE SYSTEM

Inventors:

STEPHEN M. BENNETT

ERIK COTA-ROBLES

STALINSELVARAJ JEYASINGH

GILBERT NEIGER

RICHARD UHLIG

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard, Seventh Floor
Los Angeles, CA 90025-1026

(408) 720-8300

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EV341060347US
Date of Deposit September 30, 2003
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Michelle Begay
(Typed or printed name of person mailing paper or fee)

Michelle Begay
(Signature of person mailing paper or fee)

MECHANISM TO CONTROL HARDWARE INTERRUPT ACKNOWLEDGEMENT IN A VIRTUAL MACHINE SYSTEM

Field

[0001] Embodiments of the invention relate generally to virtual machines, and more specifically to controlling hardware interrupt acknowledgement in a virtual machine system.

Background of the Invention

[0002] In a typical computer system, devices request services from system software by generating interrupt requests, which are propagated to an interrupt controller via multiple interrupt request lines. Once the interrupt controller identifies an active interrupt request line, it may send an interrupt signal to the processor. In response, the processor determines whether the software is ready to receive the interrupt. If the software is not ready to receive the interrupt, the interrupt is held in a pending state until the software becomes ready. Once the software is determined to be ready, the processor performs an interrupt acknowledgment cycle on the processor bus to request that the interrupt controller report which of the pending interrupts is of the highest priority. The interrupt controller prioritizes among the various interrupt request lines and identifies the highest priority interrupt request to the processor. The processor uses this interrupt identifier, known as the interrupt vector, to search an interrupt descriptor table (IDT) for an interrupt

descriptor pointing to code for handling this interrupt and then jumps to the handler code.

[0003] In a conventional operating system (OS), all the interrupts are controlled by a single entity known as an OS kernel. In a virtual machine system, a virtual-machine monitor (VMM) should have ultimate control over various operations and events occurring in the system to provide proper operation of virtual machines and for protection from and between virtual machines. To achieve this, the VMM typically receives control when guest software accesses a hardware resource or when certain events such as an interrupt or an exception occur. In particular, when a system device generates an interrupt, control may be transferred from the virtual machine to the VMM.

Brief Description of the Drawings

[0004] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0005] **Figure 1** illustrates one embodiment of a virtual-machine environment, in which the present invention may operate;

[0006] **Figure 2** is a flow diagram of one embodiment of a process for controlling interrupt acknowledgement in a virtual machine system;

[0007] **Figure 3** is a block diagram of one embodiment of a system for processing interrupts in a virtual machine system; and

[0008] **Figure 4** is a flow diagram of one embodiment of a process for processing interrupts in a virtual machine system.

Description of Embodiments

[0009] A method and apparatus for controlling external interrupts in a virtual machine system are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention can be practiced without these specific details.

[0010] Some portions of the detailed descriptions that follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer system's registers or memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0011] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically

stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or the like, may refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer-system memories or registers or other such information storage, transmission or display devices.

[0012] In the following detailed description of the embodiments, reference is made to the accompanying drawings that show, by way of illustration, specific embodiments in which the invention may be practiced. In the drawings, like numerals describe substantially similar components throughout the several views. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the present invention. Moreover, it is to be understood that the various embodiments of the invention, although different, are not necessarily mutually exclusive. For example, a particular feature, structure, or characteristic described in one embodiment may be included within other embodiments. The following detailed description is, therefore, not to be taken in a limiting sense, and the

scope of the present invention is defined only by the appended claims, along with the full scope of equivalents to which such claims are entitled.

[0013] Although the below examples may describe interrupt acknowledgement control in the context of execution units and logic circuits, other embodiments of the present invention can be accomplished by way of software. For example, in some embodiments, the present invention may be provided as a computer program product or software which may include a machine or computer-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. In other embodiments, steps of the present invention might be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

[0014] Thus, a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Disc, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, a transmission over the Internet, electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.) or the like.

[0015] Further, a design may go through various stages, from creation to simulation to fabrication. Data representing a design may represent the design in a number of manners. First, as is useful in simulations, the hardware may be represented using a hardware description language or another functional description language. Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. Furthermore, most designs, at some stage, reach a level of data representing the physical placement of various devices in the hardware model. In the case where conventional semiconductor fabrication techniques are used, data representing a hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. In any representation of the design, the data may be stored in any form of a machine-readable medium. An optical or electrical wave modulated or otherwise generated to transmit such information, a memory, or a magnetic or optical storage such as a disc may be the machine readable medium. Any of these mediums may "carry" or "indicate" the design or software information. When an electrical carrier wave indicating or carrying the code or design is transmitted, to the extent that copying, buffering, or re-transmission of the electrical signal is performed, a new copy is made. Thus, a communication provider or a network provider may make copies of an article (a carrier wave) embodying techniques of the present invention.

[0016] **Figure 1** illustrates one embodiment of a virtual-machine environment 100, in which the present invention may operate. In this embodiment, bare platform hardware 116 comprises a computing platform, which may be capable, for example, of executing a standard operating system (OS) or a virtual-machine monitor (VMM), such as a VMM 112.

[0017] The VMM 112, though typically implemented in software, may emulate and export a bare machine interface to higher level software. Such higher level software may comprise a standard or real-time OS, may be a highly stripped down operating environment with limited operating system functionality, may not include traditional OS facilities, etc. Alternatively, for example, the VMM 112 may be run within, or on top of, another VMM. VMMs may be implemented, for example, in hardware, software, firmware or by a combination of various techniques.

[0018] The platform hardware 116 can be of a personal computer (PC), mainframe, handheld device, portable computer, set-top box, or any other computing system. The platform hardware 116 includes a processor 118, memory 120 and one or more interrupt sources 128.

[0019] Processor 118 can be any type of processor capable of executing software, such as a microprocessor, digital signal processor, microcontroller, or the like. The processor 118 may include microcode, programmable logic or hardcoded logic for performing the execution of method embodiments of the present invention. Though **Figure 1** shows only one such processor 118, there may be one or more processors in the system.

[0020] Memory 120 can be a hard disk, a floppy disk, random access memory (RAM), read only memory (ROM), flash memory, any combination of the above devices, or any other type of machine medium readable by processor 118. Memory 120 may store instructions and/or data for performing the execution of method embodiments of the present invention.

[0021] The one or more interrupt sources 128 may be, for example, input-output (I/O) devices (e.g., network interface cards, communication ports, video controllers, disk controllers) on system buses (e.g., PCI, ISA, AGP), devices integrated into the chipset logic or processor (e.g., real-time clocks, programmable timers, performance counters), or any other source of interrupts.

[0022] The VMM 112 presents to other software (i.e., "guest" software) the abstraction of one or more virtual machines (VMs), which may provide the same or different abstractions to the various guests. **Figure 1** shows two VMs, 102 and 114. The guest software running on each VM may include a guest OS such as a guest OS 104 or 106 and various guest software applications 108 and 110. Each of the guest OSs 104 and 106 expect to access physical resources (e.g., processor registers, memory and I/O devices) within the VMs 102 and 114 on which the guest OS 104 or 106 is running and to handle various events including interrupts generated by system devices.

[0023] An interrupt may need to be handled by a currently operating VM, the VMM 112 or a VM that is not currently operating. If the interrupt is to be handled by the currently-operating VM, control remains with this VM,

and the interrupt is delivered to this VM if it is ready to receive interrupts (as indicated, for example, by an interrupt flag in a designated processor register). If the interrupt is not to be handled by the currently operating VM, control is transferred to the VMM 112. The transfer of control from guest software to the VMM 112 is referred to herein as a VM exit. After receiving control following the VM exit, the VMM 112 may perform a variety of processing, including, for example, acknowledging and handling the interrupt, after which it may return control to guest software. If the VMM does not handle the interrupt itself, it may facilitate delivery of the interrupt to a VM designated to handle the interrupt. The transfer of control from the VMM to guest software is referred to herein as a VM entry.

[0024] In one embodiment, the processor 118 controls the operation of the VMs 102 and 114 in accordance with data stored in a virtual machine control structure (VMCS) 124. The VMCS 124 is a structure that may contain state of guest software, state of the VMM 112, execution control information indicating how the VMM 112 wishes to limit or otherwise control operation of guest software, information controlling transitions between the VMM 112 and a VM, etc. When a VM exit occurs, components of the processor state used by guest software are saved to the VMCS 124, and components of the processor state required by the VMM 112 are loaded from the VMCS 124. When a VM entry occurs, the processor state that was saved at the VM exit is restored using data stored in the VMCS 124, and control is returned to guest software.

[0025] In one embodiment, the VMCS 124 is stored in memory 120. In another embodiment, the VMCS 124 is stored in the processor 118. In some embodiments, multiple VMCS structures are used to support multiple VMs.

[0026] The processor 118 reads information from the VMCS 124 to determine the execution environment of the VM and to constrain its behavior. In one embodiment, the execution control information stored in the VMCS contains an interrupt control indicator that specifies whether an interrupt generated by a system device during the operation of a VM is to cause a VM exit. Alternatively, the interrupt control indicator may reside in the processor 118, a combination of the memory 120 and the processor 118, or in any other storage location or locations.

[0027] In one embodiment, the VMM 112 sets the value of the interrupt control indicator before requesting a transfer of control to the VM 102 or 114. Alternatively, each of the VMs 102 and 114 is associated with a different interrupt control indicator that is set to a predefined value or changed during the life of the VM.

[0028] If the processor 118 determines that the pending interrupt is to generate a VM exit, the processor 118 then further decides whether this interrupt needs to be acknowledged prior to performing the VM exit. The interrupt acknowledgement involves generating an interrupt acknowledge cycle on a processor bus. In an embodiment, as part of the interrupt acknowledgement cycle, the processor 118 retrieves an identifier of the interrupt (e.g., an interrupt vector) from the interrupt controller. In one

embodiment, the determination as to whether the interrupt needs to be acknowledged depends on the current value of an interrupt acknowledge indicator. In one embodiment, the interrupt acknowledge indicator is stored in the VMCS 124 (e.g., as part of the execution control information). Alternatively, the interrupt acknowledge indicator may reside in the processor 118, a combination of the memory 120 and the processor 118, or in any other storage location or locations.

[0029] In one embodiment, the interrupt acknowledge indicator is controlled by the VMM 112. In one embodiment, the VMM 112 sets the interrupt acknowledge indicator prior to invoking a VM for the first time. For example, the VMM 112 may set the interrupt acknowledge indicator to an acknowledge value if the VMM 112 has to decide whether to handle an interrupt itself or deliver it to a specific VM based on the interrupt identifier. Alternatively, the VMM 112 may set the interrupt acknowledge indicator to a non-acknowledge value if, for example, one of the VMs 102 and 114 is designated to handle all the interrupts generated by the interrupt sources 128, and the VMM 112 always invokes this designated VM when a VM exist caused by an interrupt occurs. In one embodiment, the interrupt acknowledge indicator is modifiable by the VMM 112. For example, the VMM 112 may decide to change the interrupt acknowledge indicator if initially the system 100 had a designated VM to handle all the interrupts but later a new VM was added that is to handle some of the interrupts generated by interrupt sources 128.

[0030] If the processor 118 determines that the interrupt is to be acknowledged before the VM exit, the processor 118 acknowledges the interrupt and then transitions control to the VMM 112. In one embodiment, prior to transitioning control, the processor 118 stores the interrupt identifier obtained as part of the interrupt acknowledgement at a storage location accessible to the VMM 112. In one embodiment, the interrupt identifier is stored in the VMCS 124 (e.g., in one of the exit information fields).

[0031] **Figure 2** is a flow diagram of one embodiment of a process 200 for controlling interrupt acknowledgement in a virtual machine system. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (such as run on a general purpose computer system or a dedicated machine), or a combination of both.

[0032] Referring to **Figure 2**, process 200 begins with processing logic recognizing an interrupt pending during the operation of guest software (processing block 202).

[0033] At processing block 204, processing logic determines that the interrupt is to cause a VM exit. In one embodiment, this determination is made based on the current value of an interrupt control indicator residing in the VMCS or any other data structure. Alternatively, all interrupts may be architecturally required to cause a VM exit, and no interrupt control indicator is needed for this determination.

[0034] Next, processing logic further determines whether the interrupt is to be acknowledged prior to transitioning control to the VMM (decision box 206). In one embodiment, the determination is made using an interrupt acknowledge indicator residing in the VMCS or any other data structure. Alternatively, all interrupts may be acknowledged, and no interrupt acknowledge indicator is needed for this determination.

[0035] If the determination made at decision box 206 is positive, processing logic acknowledges the interrupt, and, in an embodiment, as part of the acknowledgement, retrieves the identifier of the interrupt (processing block 208) and stores this identifier at a storage location accessible to the VMM (processing block 210). In one embodiment, processing logic stores the interrupt identifier in the VMCS (e.g., in one of the exit information fields). A VM exit is then generated (processing block 212).

[0036] If the determination made at decision box 206 is negative, processing logic does not perform processing blocks 208 and 210 and goes directly to processing block 212.

[0037] Accordingly, with the use of process 200, performance is improved and functionality of the VMM is simplified. In particular, in systems that do not provide for interrupt acknowledgement prior to a VM exit, interrupts are blocked following every VM exit (e.g., by setting an interrupt flag in a designated processor register). As a result, in an embodiment, in order to determine the vector of the interrupt, the VMM has to unblock interrupts (e.g., by executing an instruction resetting the interrupt

flag). The processor then acknowledges the interrupt at the interrupt controller, retrieves the interrupt identifier, searches a redirection structure (e.g., the interrupt-descriptor table (IDT) in the instruction set architecture (ISA) of the Intel® Pentium® 4 (referred to herein as the IA-32 ISA)) for an entry associated with the interrupt, extracts from this entry a descriptor of a handler associated with this interrupt, and jumps to the beginning of the appropriate VMM handler code using the descriptor. The VMM can then handle the interrupt appropriately since it can identify the interrupt source based on which handler was invoked by the processor. Alternatively, in another embodiment, following the VM exit, a VMM may perform a series of accesses to the interrupt controller (e.g., I/O accesses) to determine the interrupt vector and to acknowledge the interrupt.

[0038] With the use of process 200, the interrupt can be acknowledged prior to the VM exit. As a result, the VMM has an immediate access to the interrupt identifying information once the VMM receives control. The VMM can then use the interrupt identifier to find the appropriate interrupt handler code or VM. Consequently, the need for the VMM to unblock the interrupts is eliminated, as well as the need for the processor to search a redirection structure, extract from a corresponding entry a descriptor of a handler associated with this interrupt, and jump to the beginning of the appropriate VMM exception handler code. This, in turn, can simplify the VMM software design and validation because the VMM no longer needs to have necessary

code and/or structures as described above (e.g., a redirection structure (e.g., the IDT), code to acknowledge interrupts using I/O operations, etc).

[0039] **Figure 3** is a block diagram of one embodiment of a system 300 for processing interrupts in a virtual machine environment.

[0040] Referring to **Figure 3**, devices 314 (e.g., I/O devices) request services from system software by generating interrupt requests, which are propagated to an interrupt controller 313 via one or more interrupt request lines 316. Once the interrupt controller 313 identifies an active interrupt request line 316, it sends an interrupt signal 310 to the CPU 302. The interrupt controller 313 may include masking and prioritization logic that is know to one skilled in the art. In an embodiment, there may be more than one interrupt signal line 310 to the CPU 302, or, alternatively, the interrupt "signal" may be delivered via a bus message or through any other communication mechanism or protocol.

[0041] In response to an active interrupt signal 310 from the interrupt controller 313, interrupt controller interface logic 304 determines which software has control over the interrupt. If the interrupt occurs during the operation of a VMM, the interrupt is managed by the VMM. Alternatively, if the operation occurs during the operation of guest software, the interrupt controller interface logic 304 determines whether the interrupt is controlled by the currently operating guest software or by the VMM. If the interrupt is controlled by the VMM, then a VM exit is generated to transfer control to the VMM.

[0042] In one embodiment, this determination depends on the current value of an interrupt control indicator 320 stored in VMCS 308 (e.g., as part of the execution control information). In one embodiment, the VMM sets the value of the interrupt control indicator 320 prior to requesting a VM entry into a specific VM for the first time and does not change the value throughout the life of the VM. For example, if the system has a certain VM that handles all interrupts, the interrupt control indicator 320 for this VM would be set to allow guest control of interrupts; the interrupt control indicator 320 for all other VMs would be set to VMM control. Alternatively, if interrupts can be handled by different VMs and/or the VMM depending on the interrupt identifiers and the VMM needs to decide which entity handles the present interrupt, then the interrupt control indicator 320 may be set to indicate VMM control for all VMs. In yet other embodiments, the value of the interrupt control indicator 320 may change during the lifetime of a particular VM.

[0043] If the interrupt control indicator 320 specifies that the interrupt is controlled by the currently operating VM, the interrupt controller interface logic 304 further determines whether the currently operating VM is ready to receive interrupts. In one embodiment, the interrupt controller interface logic 304 makes this determination upon consulting an interrupt flag 306 that can be updated by guest software when the state of guest software's ability to accept interrupts changes. For example, in the IA-32 ISA, the EFLAGS register contains the IF interrupt flag bit, which, in part, controls whether an interrupt will be delivered to the software (other factors may block interrupts in the IA-

32 ISA and these factors must be considered in determining if an interrupt may be delivered). In an embodiment, the interrupt flag 306 resides in the CPU 302 outside or inside the interrupt controller interface logic 304.

[0044] If the interrupt controller interface logic 304 determines that the guest software is ready to receive the interrupt, it acknowledges the interrupt request at the interrupt controller 313 by an interrupt acknowledgment cycle, for which, in an embodiment, the interrupt controller 313 returns the identifier of the interrupt (referred to as an interrupt vector) that has the highest priority. Based on the interrupt vector, the interrupt controller interface logic 304 searches the IDT of this VM for an entry associated with the interrupt, extracts from this entry a descriptor of a handler associated with this interrupt, and jumps to the beginning of the appropriate VM exception handler code using the descriptor. In anoter embodiment, there is a single interrupt handler and no interrupt identifier is required. Otherwise, if the VM is not currently ready to receive interrupts, the interrupt is held in a pending state until the VM becomes ready.

[0045] If the interrupt control indicator 320 specifies that the interrupt is controlled by the VMM (i.e., a VM exit is to be generated), then, in one embodiment, the interrupt controller interface logic 304 consults an interrupt transition flag referred to herein as a monitor interrupt flag (MIF) 322. The MIF 322 behaves in a manner that is analogous to the interrupt flag 306, indicating whether interrupts are allowed to cause transitions to the VMM. In one embodiment, the MIF 322 resides in the VMCS 308 and is controlled by

the VMM. In another embodiment, the MIF 322 resides in a machine register or in memory. If the MIF 322 indicates that interrupts are blocked, no VM exit will be generated until the MIF 322 is changed to unblock interrupts. The VMM may set the MIF 322 to a blocking value if, for example, the currently operating VM performs a time critical task and the VMM does not want any VM exits to occur to avoid reducing the performance of this VM.

[0046] In another embodiment, the interrupt controller interface logic 304 does not consult the MIF 322 and decides whether the interrupt is to cause a VM exit based only on the interrupt control indicator 320.

[0047] If the interrupt controller interface logic 304 determines that the interrupt is to cause a VM exit, the interrupt controller interface logic 304 further consults an interrupt acknowledgement indicator 324. The interrupt acknowledgement indicator 324 specifies whether the interrupt is to be acknowledged prior to a VM exit. In one embodiment, the interrupt acknowledgement indicator 324 resides in the VMCS 308 and is controlled by the VMM. In another embodiment, the interrupt acknowledgement indicator 324 resides in a machine register or in memory.

[0048] If the interrupt acknowledgement indicator 324 specifies that no acknowledgement is required prior to a VM exit, the interrupt controller interface logic 304 generates a VM exit. The VMM may then, for example, set the interrupt control indicator 320 to a VM control value and request a transition of control to a VM designated to handle all the interrupts. In another example, the VMM may determine the interrupt vector itself (e.g., by

accessing an appropriate register of the interrupt controller 312 to read the vector of the interrupt).

[0049] If the interrupt acknowledgement indicator 324 specifies that the interrupt is to be acknowledged, the interrupt controller interface logic 304 acknowledges the interrupt at the interface controller 312, and, in an embodiment, retrieves the interrupt vector value from the interrupt controller 312 and stores the interrupt vector value in the interrupt vector field 326 of the VMCS. A VM exit is then generated. Upon receiving control, the VMM may determine how the interrupt should be handled (e.g., by vectoring to a handler in the VMM, by invoking the appropriate VM to handle the interrupt, etc.). In an embodiment, this determination may be based upon the interrupt vector 326 from the VMCS 308.

[0050] **Figure 4** is a flow diagram of one embodiment of a process 400 for processing interrupts in a virtual machine system. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (such as run on a general purpose computer system or a dedicated machine), or a combination of both.

[0051] Referring to **Figure 4**, process 400 begins with processing logic identifying the presence of a pending interrupt during the operation of guest software (processing block 402). Next, processing logic determines whether the interrupt is to be controlled by a VMM or the guest software using an interrupt control indicator (decision box 404). If the interrupt control indicator

specifies that the interrupt is to be controlled by the guest software, processing logic further determines whether the guest software is ready to receive interrupts (decision box 406). In one embodiment, this determination is made by consulting an interrupt flag (e.g., the IF bit in the EFLAGS register). If the determination is positive, processing logic delivers the interrupt to the guest software (processing block 408). The delivery of the interrupt to the guest software includes acknowledging the interrupt with the interrupt controller, retrieving the interrupt vector and passing control to the guest at the appropriate handler based on the vector. Otherwise, the interrupt is held pending (processing block 410) until the guest software becomes ready to handle interrupts.

[0052] If the interrupt is controlled by the VMM, in one embodiment, processing logic determines whether a monitor interrupt flag (MIF) is set to a blocking value (decision box 412). If so, the interrupt is held pending (processing block 414) until the MIF is changed to an unblock value. If not, processing logic further determines whether the interrupt is to be acknowledged using an interrupt acknowledge indicator (decision box 416). If not, processing logic does not acknowledge the interrupt and goes directly to processing block 424 to generate a VM exit.

[0053] If the interrupt is to be acknowledged, processing logic acknowledges the interrupt at the interrupt controller (processing block 418). In some embodiments, processing logic retrieves the interrupt vector from the interrupt controller (processing block 420) and stores the interrupt vector in

the VMCS (processing block 422). Finally, processing logic generates a VM exit (processing block 424).

[0054] Thus, a method and apparatus for handling interrupts in a virtual machine system have been described. It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reading and understanding the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.